Chap.6 자료형(Data Type)

6.1 Data Type & Type Declaration

* Data Type : Object

  \+ Object instance를 Create,

  연산 { build-up, destroy, modify, pick-apart

* built-in data type
  integer
  real
  character
  logical (boolean)

  cf) composite data-type

  P.151 <표 6.1> built-in data type

cf) Typing Mechanism
: 새로운 자료형(data type)을 제공하고, variable을 어느 한정된 data type으로 선언해 주는 facility

* data type declaration 의 유용한 점
  ① static type checking 가능
  ② 신뢰성 증가, 프로그램 읽기 쉽게

cf) strongly type
: data type 에 관한 모든 특성들이 compile time 에 확정되는 Programming Language
장점) reliability, maintainability, readability
단점) data type 에 관한 정보를 미리 모두 정해야 함

cf) scalar type
: data type의 domain이 상수값으로만 구성
In pascal { integer, real, boolean, char

structured type
: element들은 field들을 가지고 있으며
이 field들은 자신의 data type을 따로 갖는다.
ex) array, record, struct

6.2 단순형

o 내장된 자료형: 단순형(수치자료형, 논리형, 문자형)

p.154 그림6.1 Pascal 자료형의 분류

p.154 그림6.2 Ada 자료형의 분류

p.155 그림6.3 Java 자료형의 분류

6.2.1 Numberic Type(수치형)

- integer
- real

: 구현될 때 기계에 dependent
장점) 기계의 내장된 코드로 변환되므로 연산 속도가 매우 빠름
단전) 다른 기계로 이동시킬 때 어려움
(기계에 따라 정밀도(precision)가 다름)

cf) polymorphism
: 다양한 객체들이 동일한 메시지에 다르게 응답하는 능력
+ : 정수 덧셈, 실수 덧셈, 행렬 덧셈

p.157 표6.2 정수-실수 수식들을 다루기위한 규칙들

cf) type cast
: expression의 data type을 explicit하게 변환

```
real j
i = (int) j
```

cf) narrowing & widening
실수 → 정수 정수 → 실수
정보 손실 유무 ’

6.2.2 논리형(Boolean Type, Logical Type)

* Boolean domain - true
  false

operation { and, or, not, implies(imp), equivalence(equiv)

x and y = if x then y else false
x or y = if x then true else y
not x = if x then false else true
x imp y = if x then y else true
x equiv y = if x then y else not y

6.2.3 문자형(Character)

<<Char 조작 필요성>>
- o variable 문자열
- o relational operator (순서)
- o 문자열 분해 / 조립

cf) SNOBOL 1964

6.3 Enumerated data types (열거 자료형)

operation : equivalence, order, assignment

In Pascal,

type months = (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);
: 12 개의 literal로 구성된 "months"라 부르는 enumerated type

① months 형 변수 선언

```
var x,y,z : months
x := Jan            "assignment"
y := Jun
if x = y then x := Nov else z := Dec
   "equivalence"
```

② order

> < <> = <= >=

pred(Jun) = May

succ(Jun) = Jul

cf) subrange
type summer mouths = Jun..Aug
== Jun, Jul, Aug

6.4 배열

o structured Data Types : 여러 자료를 묶어서 하나의 단위로 처리
- array : 위치로 원소를 식별하는 homogeneous data 모임
- record : 이름으로 원소를 식별하는 heterogeneous data 모임

o Array
- 이름
- Array dimension
- element type
- index type / range

o index { 상수(정수) [lb..ub] , lb ≤ ub, ub-lb+1
expression

eg) FORTRAN, PL/I, Ada
A : 5 * 3 * 7 => A(5, 3, 7)

eg) ALGOL, Pascal => A[5, 3, 7]

o static array(정적배열)
- index 범위가 정적으로 바인딩되고 기억장소 할당이 정적(실행시간전)으로 이루어지는 배열
- 효율성(동적 기억장소 할당 및 회수가 필요하지 않음)
- 정적변수(static variable)

예: FORTRAN 77

o fixed stack-dynamic array(고정 스택-동적 배열)
- index 범위가 정적으로 바인딩되지만, 기억장소 할당이 실행시간중 선언을 갖는 블록이 활성화될 때 이루어지는 배열
- 기억장소 공간의 효율성
- semi-static variable

예: Pascal 프로시져와 C 함수에서 선언된 배열(C에서 static 명세가 없는 경우)

o stack-dynamic array(스택-동적 배열)
- index 범위가 동적으로 바인딩되고 기억장소 할당이 동적으로(실행시간중) 이루어지는 배열
- index 범위가 바인딩되고 기억장소가 할당되면, 바인딩과 할당은 그 변수의 존속기간동안에 고정됨
- 유연성(flexibility)
- 배열크기는 배열이 사용직전에 생성되기 전까지는 알려질 필요가 없음
- semi-dynamic variable

예: ALGOL 68, Ada

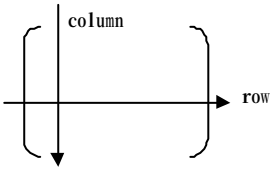
```
GET Length;
.......
declare
    A: array(1..Length) of FLOAT;
begin
    .......
end;
```

o heap-dynamic array
- index 범위의 바인딩과 기억장소 할당이 동적으로(실행시간중) 이루어지고, 이것이 그 배열의 존속기간동안에 여러번 변경될 수 있는 배열
- 배열은 프로그램 실행 중 필요한 기억장소 공간이 변화함에 따라 커지거나 줄어듬

-dynamic variable, heap variable

예: Fortran I
Ada
C : malloc, free
C++ : new, delete

eg) row major
column major



p.173 그림 6.6 PL/I에서의 slice

p.175 배열의 설계 방법

6.5 Associative Array(연상 배열)

o key라 불리는 값들에 의해서 접근되는 순서를 갖지않는 데이터 원소들의 집합체

o p.176 Perl의 hash

```
%salaries = ("HONG" => 1200, "WON" => 2000,
             "KIM" =>1500, "LEE" => 2500);

$salaries{"WON"} => 2000;     참조

delete $salaries{"LEE"};      삭제

@salaries = ( );    공 hash
```

6.6 Records

o type stock =

```
record
   name : array(1..30) of char
   price : array(1..365) of real
   dividend : real
   volume : array[1..365] of integer
   exchange : (hyse, amex, nasdaq)
end
```

var ibm : stock

=> "field 참조 방법 “

o knuth : functional notation { name (ibm), price(ibm), ⋮

o "." 표기법 : { ibm.name, ⋮

o variant-part : 자료형에 따른 몇개의 택일을 가능하게 함

type option = (sixmonths, ninemonths)

p. 181

```
case x : option of
     discriminent

   sixmonths : (exprice : real);
   nine months : (nuprice : integer)

end
```

6.7 Pointer Type

o pointer : 어떤 객체에 대한 reference (location)
o pointer variable : 객체를 referencing 하기 위한 주소를 값으로 가지는 identifier
이용) ① 실행때까지 생성된 항목 개수가 정해지지 않았을 때
② 필요할 때 마다 : memory allocation
불필요할 때 마다 : free

문제점) P184 중간

pointer 표기의 예) P189 (표 6.5)

6.8 자료형의 변환 (Type Coercion(강제))

: 주어진 자료형의 값을 다른 자료형의 값으로 변환
cf) static type checking
cf) dynamic type checking

o Type Coercion
widening : 정수 → 실수
narrowing : 실수 → 정수 truncation, rounding

6.9 자료형의 동치 pp194-198

o type compatibility : 객체의 형이 특정 context에서 정당한지를 결정하는 semantics rules을 말한다

① [name equivalence]
두 변수를 함께 선언하거나 동일 식별자 이름을 사용해서 선언하면 두 변수는 동일형으로 인식되며 반대의 경우도 성립함
x, y

② [structural equivalence]
자료형의 구성 요소가 모든 측면에서 같다면 두 변수는 동일형으로 인식되며 반대의 경우도 성립함
w, x, y, z

```
type  T = array[1..100] of integer
var   x, y : array[1..100] of integer
      z : array[1..100] of integer
```

w : T

③ declaration equivalence : 같이 기술되어 선언된 경우만 compatibility

```
type stock =
        record
          name
          ...
        end
var a, b : stock        → declaration equivalence
          c : stock
```